L'objectif de ce projet est de programmer un bot pour le jeu **A Song of Link and Wire** présenté dans la Section 1.

Instructions

Ce projet est à réaliser impérativement en binôme.
Seules les bibliothèques spécifiquement mentionnées dans le sujet sont autorisées.

1 A Song of Link and Wire

Quelque part entre la disparition des osselets et le raz-de-marée *Candy Crush*, les étudiants et étudiantes de l'UPEC jouaient à la *Course wikipédienne* entre (et parfois pendant) les cours. Ce jeu aux règles simples se déroulait de la manière suivante : on commençait par choisir deux événements, personnages historiques ou lieux célèbres, si possible ayant peu à voir l'un avec l'autre. L'objectif était alors, en partant la page wikipédia du premier, de rejoindre celle du second, en suivant le plus petit nombre possible de liens hypertexte.

Nous allons ici étudier une variante de ce jeu : **A Song of Link and Wire**, qui sur déroule sur le wiki Game of Throne, situé à l'adresse `https://iceandfire.fandom.com/wiki/`. Notre premier objectif est d'implémenter un programme répondant au problème suivant :

> **Entrée :** Deux chaînes de caractères `source` et `cible` représentant des pages du wiki, i.e. telles que les URL `"https://iceandfire.fandom.com/wiki/"+source` et `"https://iceandfire.fandom.com/wiki/"+cible` sont valides.
>
> **Sortie :** Ou bien
>
> - une liste `l` de chaînes de caractères telle que
>   - `l[0]==source`
>   - `l[len(l)-1]==cible`
>   - pour tout `i` entre `0` et `len(l)-2`, `l[i]` est une page du wiki, et possède un lien vers la page `l[i+1]`,
> - ou bien `None` s'il n'y a pas de chemin de `source` vers `cible` dans le wiki.

Par exemple, pour les deux pages `"Dorne"` et `"Rhaego"` (le fils mort-né de Daenerys et du Khal Drogo), la sortie sera
`["Dorne","House_Targaryen","Rhaego"]`
Il existe des paires de pages pour lesquelles plusieurs chemins de plus courte longueur existent : dans ce cas, n'importe lequel de ces chemins est une réponse valable.

2 Lister les liens

Rappelons que les pages du wiki seront représentées par des chaînes de caractères.
Par exemple, la page `https://iceandfire.fandom.com/wiki/Petyr_Baelish` sera représentée par la chaîne `"Petyr_Baelish"`.

Question 1 Écrire une fonction `liste_liens()` qui attend en argument (une chaîne de caractères représentant) une page `page` du wiki, et qui renvoie la liste des pages du wiki vers lesquelles pointe `page`. On ne prendra évidemment en compte que les liens contenus dans le corps de `page`, et pas les liens situés à la périphérie : on étudiera donc la structure des pages html du wiki via l'inspecteur de son navigateur favori.

Indications : Pour récupérer le fichier html à l'adresse `adresse`, on pourra faire un appel à `requests.get(adresse)`, après avoir installé et importé la bibliothèque `requests`. Pour traiter le fichier html, on utilisera la bibliothèque `BeautifulSoup`, dont on trouvera la documentation et les instructions d'installation à l'adresse suivante.

3 Construction du graphe

Il ne serait pas raisonnable, tant du point de vue de la consommation énergétique que de la latence, de récupérer les pages html du wiki à chaque fois qu'on exécute le programme. On va donc stocker une bonne fois pour toutes le graphe du wiki dans un fichier local.

La manière la plus naturelle de représenter le wiki est via un dictionnaire, dont les paires `clef:valeur` auront le sens suivant :

- `clef` est une page du wiki
- `valeur` est la liste des pages du wiki vers lesquelles pointe `clef`.

Question 2 Implémenter une fonction `svg_dico()` qui

- attend en premier argument un dictionnaire `dico` dont les clefs sont des chaînes de caractères, et les valeurs des listes de chaînes de caractères,
- attend en deuxième argument un nom de fichier `fichier`, et
- écrit le contenu de `dico` dans `fichier`, à raison d'une ligne par `clef:valeur`.

Question 3 Implémenter une fonction `chg_dico()` qui attend en argument un nom de fichier (qu'on supposera résulter d'un appel à `svg_dico()`), et renvoie un dictionnaire représentant le contenu du fichier.

Avant de passer à la question suivante, gourmande en temps et en ressources, **on s'assurera d'avoir bien testé les deux fonctions précédentes**.

Question 4 En implémentant un parcours de graphe (on pourra se rafraîchir ici la mémoire sur le parcours en largeur) depuis la page de Petyr Baelish, sauvegardez dans un fichier une représentation du wiki.

Pour toute la suite du projet, c'est sur cette représentation qu'on travaillera, sans jamais retourner sur le wiki. Il est recommandé de dupliquer ce fichier pour ne pas devoir recommencer l'opération en cas de mauvaise manipulation.

4 Variations sur le thème du plus court chemin

On est maintenant parés pour attaquer le cœur du projet : la recherche d'un plus court chemin entre deux pages dans le wiki.

Question 5 À l'aide d'un nouveau parcours en largeur, écrire une fonction `plus_court_chemin()` répondant aux spécifications présentées dans l'encart de la Section 1.

On vérifiera bien que pour les deux pages `"Dorne"` et `"Rhaego"`, cette fonction renvoie la liste `["Dorne","House_Targaryen","Rhaego"]`.

On va maintenant corser les choses, et modifier la topologie du wiki. Plutôt que de considérer, comme on l'a fait jusque-là, que tous les liens ont le même poids, on va pondérer différemment chaque lien.

Plus précisément, on considère que suivre un lien vers la page `cible` a un coût donné par la formule `nombre_caractères(cible) + nombre_voyelle(cible)`. Autrement dit, le coût est égal à la longueur de `cible`, où chaque voyelle (a,e,i,o,u,y, en minuscule ou majuscule) compte double.

Question 6 En implémentant l'algorithme de Dijkstra, écrire une fonction `pcc_voyelles()` qui répond au même problème, mais cette fois en cherchant un chemin de poids minimal vis-à-vis de la nouvelle métrique - le poids d'un chemin étant défini comme la somme des poids des liens qui le composent.